CSCI 620 – Intro to Big Data

Assignment #4 – Document Databases

Name: *Yuvraj Singh*

1a) Creating the Database

The database was created by installing MongoDB and then installing the *PyMongo* dependency for Python. *PyMongo* handles creating the database along with the connection if they do not exist when data is inserted. The *def __init__* method of the *class Mongo* in *Mongo.py* handles all of this. All it needs is the database location which needs to be stored in the config file along with the database name.

1b) Preparing the Data and Populating the Database

The data for the database was taken from the CSVs generated from assignment #2 since all that data is already preprocessed and cleaned. Unfortunately, those CSVs were generated to be used with the COPY command with Postgres, so the data needed to be combined and formatted a bit differently to work with the collection schemas for this database. Copy over the data folder from assignment 2 and place it in the root folder where the rest of the files for this assignment should be. By calling the *def insert_docs* method in *Mongo.py*, the Movies and Members collections are populated with the newly formatted data. To format the data properly and to populate the database, it took approximately 15 minutes.

```
(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4> cd c:\Users\Yuvraj\Desk
 -- c:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4\mongo.py "
It took 54.217707700000005 seconds to format genre via numpy
It took 29.696191499999998 seconds to format title_director via numpy
It took 21.1964354 seconds to format title_writer via numpy
It took 15.054058399999988 seconds to format title_producer via numpy
It took 41.97632610000001 seconds to merge everything in
It took 79.8631905 seconds to covert the dataframe to json and removed keys with None
It took 186.7880983 seconds to create a dict with each actor and their role to its respective movie
It took 3.339513600000032 seconds to merge actors and their roles to each title
It took 212.2355379 seconds to load the json file to the collection
It took 37.65413860000001 seconds to covert the dataframe to json and removed keys with None
It took 200.9200816 seconds to load the json file to the collection
```

2) Custom Queries

For the second part of the assignments, queries had to be created for certain lookups in the database. Each query was then created and ran in the *def queries* method in *mongo.py*. After each query is ran, its execution plan is then generated and stored into a json file that corresponds to each query. I ran into some issues when performing queries that involved actors. When I would perform a *$lookup* on the actors array for the actors id, it wouldn't perform the lookup properly and thus return nothing. Every step prior to the *$lookup* on *'localField': 'actors.actor'* would run as expected but I couldn't seem to have figured out how to correctly join Members to Actors. The same issue was apparent when performing the fifth query. The query was able to return movies that matched Sci-Fi and the director was James Cameron, but unable to filter and get movies that Sigourney Weaver was an actor. These invalid queries are in the *def queries_err* method. Other than the incompletion for those two queries, the others worked as expected. The query times are printed out along with some of the returned data when running the *def queries* method.

```
It took 6.916389499999999 seconds to for query (2.2)
{'_id': 1861174, 'name': 'Dominic Gillette', 'count': 69}
{'_id': 8230849, 'name': 'Ryan Gill', 'count': 79}
It took 34.8690759 seconds to for query (2.3)
{'_id': None, 'avgRuntime': 103.14285714285714}
It took 20.3283876 seconds to for query (2.4)
{'_id': [438506], 'name': ['Shobha Kapoor'], 'count': 671}
{'_id': [8467983], 'name': ['Saibal Banerjee'], 'count': 439}
{'_id': [2255484], 'name': ['Rashmi Sharma'], 'count': 377}
{'_id': [1192808, 1421119], 'name': ['Mahendra Soni', 'Shrikant Mohta'], 'count': 232}
{'_id': [11482207], 'name': ['Vaidehi Ramamurthy'], 'count': 215}
{'_id': [1637844], 'name': ['Deeya Singh'], 'count': 203}
{'_id': [1754670], 'name': ['Nilgun Sagyasar'], 'count': 185}
{'_id': [2342441], 'name': ['Lisa Lew'], 'count': 183}
{'_id': [10695414], 'name': ['R. Ramesh Babu'], 'count': 182}
{'_id': [702732, 2562305, 2830020], 'name': ['Federico S. Quadrani', 'Russell McCarroll', 'Eric Ortner'], 'count': 164}
```

3) Query Explanations

For each query, a JSON file is created that contains the Explain statement from MongoDB. (Queries 1 and 5 are omitted since they were not fully functional)

Query 2.2

The first stage of this query is a cursor planner and then a projection on producer.id, producers._id, producers.name, and _id. The next stage consists of a COLLSCAN on startYear and filtering startYears equal to 2017. This stage represents the $match on startYear for Movies in the query. The next stage is unwinding Genre, and then performing a match on genre = 'Talk-Show'. After performing the matching, the next stage is joining Members via $lookup with the producer_id field. After joining Members, a match is performed on producers whose names contains "Gill". Then, the producer_id and name are grouped and counted, which is then filtered using a match to those groups whose counts were greater than 50.

Query 2.3

The first stage of this query begins with a cursor planner and then a projection on runtimeMinutes, writer_id, writers.deathYear, writers.name, and _id. A COLLSCAN is then performed, filtering titleType = movie. After the scan is complete, a lookup is done to join Members to writer_id and then a match is performed on writers whose name contains "Bhardwaj" and their deathYear is null. When the matching is complete, the ids are grouped together, and the average runtime minutes is then computed with the results from the match.

Query 2.4

The first stage begins with a cursor planner and then a projection on producer_id, producers._id, producers.deathYear, producers.name, and _id. After the projection is finished, a COLLSCAN begins

and filters Movie documents with runtimeMinutes greater than or equal to 120. When the scan is complete, a lookup is performed to join Members to producer_id. A match stage then occurs to find the producers who do not have a deathYear. Once that matching is complete, the results are then grouped by producers._id and is then counted. The final stage sorts the results by count of whichever producer_id occurred the most.

4) Creating Indexes

Indexes were created for each query and then was tested to see if it helped with performance. To create the indexes and check the performance, run *def queries* first to get a base line of how the performance is without indexes, and then run *def create_indexes* to see the performance with the indexes. (Queries 1 and 5 are omitted since they were not fully functional)

```
It took 6.916389499999999 seconds to for query (2.2)
{'_id': 1861174, 'name': 'Dominic Gillette', 'count': 69}
{'_id': 8230849, 'name': 'Ryan Gill', 'count': 79}
It took 34.8690759 seconds to for query (2.3)
{'_id': None, 'avgRuntime': 103.14285714285714}
It took 20.3283876 seconds to for query (2.4)
{'_id': [438506], 'name': ['Shobha Kapoor'], 'count': 671}
{'_id': [8467983], 'name': ['Saibal Banerjee'], 'count': 439}
{'_id': [2255484], 'name': ['Rashmi Sharma'], 'count': 377}
{'_id': [1192808, 1421119], 'name': ['Mahendra Soni', 'Shrikant Mohta'], 'count': 232}
{'_id': [11482207], 'name': ['Vaidehi Ramamurthy'], 'count': 215}
{'_id': [1637844], 'name': ['Deeya Singh'], 'count': 203}
{'_id': [1754670], 'name': ['Nilgun Sagyasar'], 'count': 185}
{'_id': [2342441], 'name': ['Lisa Lew'], 'count': 183}
{'_id': [10695414], 'name': ['R. Ramesh Babu'], 'count': 182}
{'_id': [702732, 2562305, 2830020], 'name': ['Federico S. Quadrani', 'Russell McCarroll', 'Eric Ortner'], 'count': 164}
```

Query 2.2

For the second query, an index was created for the Movies collection and it was on ‘Genre’. An index was also created for the Members collection and it was on ‘Name’. In the execution plan, we can see that there are matches on ‘Genre’ for the Movies collection and for the Members collection, each name had to be checked to see if it contained “Gill”, which is why these were chosen to create the indexes on.

After creating the indexes, and checking the query time, we can see a small performance boost. Originally it took about 6.916 seconds and then with the indexes, it took about 6.78 seconds.

```
(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4> cd c:
 -- c:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4\mongo.py "
It took 6.736860600000001 seconds to for query (2.2)

(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4> cd c:
 -- c:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4\mongo.py "
It took 6.7808074000000005 seconds to for query (2.2)

(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4>
```

Query 2.3

For the third query, an index was created for the Movies collection and it was on ‘titleType’. An index was also created for the Members collection and it was on ‘Name’. The reason why these fields for indexes was because matches had to be performed on titleType, and each name had to be checked to see if it included a substring. After creating the indexes, and checking the query time, we can see another small performance boost. Originally it took about 34.869 seconds and then with the indexes, it took about 33.79 seconds.

```
(bigdata) C:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4> cd c:\Users\
 -- c:\Users\Yuvraj\Desktop\Fall_2020\CSCI-620\assignments\assignment4\mongo.py "
It took 31.918789100000005 seconds to for query (2.3)
```

Query 2.4

For the fourth query, an index was created for the Movies collection and it was on ‘Genre’ and ‘startYear’. An index was also created for the Members collection and it was on ‘Name’. For Movies, genre and startYear was chosen because matches had to be done for those fields. For Members, name was chosen since it was part of the group clause and deathYear since a match had to be performed on it to check that there is not a field of deathYear for each producer. After creating the indexes, and checking the query time, we can see that it did not change much as for performance wise. Originally it took about 20.32 seconds and then with the indexes, it took about 20.2195 seconds so not much.